Multi-agent control and intelligent sensor allocation with Reinforcement Learning and Genetic Programming

Final Report 0001AD
Icosystem Corporation, 3 February 2003

Overview

This document describes work performed under STTR contract N00014-02-M-0266 by Icosystem Corporation, in collaboration with MIT AI Lab staff, for the months of December, 2002 and January, 2003. This is the fourth and final report to describe our work.

In the sixth and seventh months of effort, we have performed the following tasks:

- Extended and refined the UGV evader-pursuer simulation tool in several ways:
    - the dynamics of the UGVs (both types) are more accurate and better able to handle navigation even in tight spaces;
    - the World Editor and the Agent-Based model are now fully integrated: users can modify the environment or the navigation points while the simulation is running, if desired;
    - the simulator allows selection of several run-time parameters, including the number of pursuers and the evader strategy;
    - the simulator can run in "batch" mode with variable random seed for Monte Carlo simulations
- Hand-designed a variety of pursuer and evader strategies
- Ran Monte Carlo simulations to test the efficacy of several strategies under varying conditions.

Design of control strategies

At the time of the previous report, both pursuers and evaders had their vehicles simulated, with low level control and obstacle avoidance. Given a point far away, they were able to plan a path to it and head there, using a combination of reactive strategies at lower levels (path execution and obstacle avoidance) and deliberative algorithms at higher levels (path planning and A* search).

Since then we have added an infrastructure for hardest part of the task, making the tactical decision of what path to take to find and capture the evader, or to best hide or flee from the pursuer. This task is greatly simplified by recognizing that, once a decision has been made, we needn't revisit it every time step. In other words, the state of the world is very similar from one time step to the next, so the best action at any time step is mostly likely the same as it was at the previous time step.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
03-02-2003	Final Report	01-07-2002 to 31-01-2003

4. TITLE AND SUBTITLE

Multi-agent Control and Intelligent Sensor Allocation with □ Reinforcement Learning and Genetic Programming

5a. CONTRACT NUMBER

N00014-02-M-0266

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Gaudiano, Paolo

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Icosystem Corporation□
10 Fawcett Street□
Cambridge, MA 02128

8. PERFORMING ORGANIZATION REPORT NUMBER

0001AD

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Office of Naval Research□
Ballston Tower One□
800 North Quincy Street□
Arlington, VA 22217-566-□
□

10. SPONSOR/MONITOR'S ACRONYM(S)

ONR

11. SPONSORING/MONITORING AGENCY REPORT NUMBER

0001AD

12. DISTRIBUTION AVAILABILITY STATEMENT

Approved for public release

13. SUPPLEMENTARY NOTES

14. ABSTRACT

This project investigated the use of decentralized strategies for control of UAV and UGV swarms. During the first few months we □ developed an agent-based model of UAVs searching for targets in a pre-determined search area. We tested a variety of control □ and navigation strategies, including some based on biological principles. Subsequently we develop a second simulator, focusing on □ the problem of a team of pursuers trying to capture an evader in a 2-D urban environment. The simulator included control strategies □ for the evader and the pursuers, as well as an interactive world editor for creation of arbitrary urban environments. We used this □ simulator to run Monte Carlo simulations, obtaining some preliminary statistics on performance of the evader and pursuer□ strategies.□
□

15. SUBJECT TERMS

Agent-based modeling, pursuer-evader, decentralized control□
□

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Paolo Gaudiano
U	U	U	UU	7	19b. TELEPONE NUMBER (*Include area code*) 671-520-1070

Therefore, each vehicle, be it a pursuer or evader, only updates its goals when there's a significant change in either its state or the world. In particular, a pursuer only makes decisions when it reaches an intersection, when it or another pursuer sees the evader, when it looses sight of the evader, or when the evader abruptly changes direction.

This strategy can greatly speed up the simulation, which allows many more strategies to be evaluated by evolution. It also allows the individual decisions to be more involved and time consuming.

Early experiments revealed that these decisions depended on the topological structure of the world. A natural way of searching this is to use a best first search, using machine learning to discover a heuristic function to evaluate the suitability of each intersection as a possible goal. This strategy has proven wildly successful in games such as checkers and backgammon. In fact, the best computer backgammon player to date, TD-Gammon, was created using exactly this strategy.

Initial experiments creating both pursuers and evaders by hand have demonstrated a variety of possible behaviors. Many interesting and useful strategies seem to be easily represented in this framework. Thus, we have a framework which is ready for evolution, and which is organized such that evolution should discover many interesting behaviors.

In our early runs we found that the performance of pursuit strategies also depends heavily on the evader strategies. For this reason we have focused more on designing various strategies for the evader. In any event, because of the complexity of the team control, the design of pursuer strategies will be best handled with evolutionary computing.

For this period of performance, we designed a fairly simple pursuer strategy, and we tested it systematically against several hand-designed evader strategies. The current pursuer strategy only relies minimally on teamwork: each pursuer sets an arbitrary "goal point" in the environment and navigates toward it. If a pursuer sees the evader (line of sight), it will head toward it, while notifying the other pursuers. If the evader disappears around a corner, the pursuer "remembers" the evader's last location for 5 seconds, and keeps heading in that direction. If after 5 seconds the evader is still not visible, and no other pursuer sees it either, the pursuer selects a new random point in the environment to which to navigate. When a pursuer is notified that another pursuer has seen the evader, it will head toward the location where the evader was seen.

Before reporting some simulation results, we turn to a detailed description of the evader strategy.

Evader strategies

We have designed and tested several evader strategies. We could group the strategies into two classes: rule-based strategies and reactive strategies. In all cases we decided to assume that the evader can only see pursuers line-of-sight, though its speed is higher than the speed of the pursuers. We felt this is a more faithful representation of a real situation in which an evader is trying to escape within an urban environment.

By *rule-based strategy* we refer to a strategy in which the evader follows a series of if-then rules whenever it reaches a waypoint or it sees a new pursuer. In contrast, a *reactive strategy* is one in which the evader treats various factors in the environment, including the presence of pursuers, as

analog input signals. The moment-by-moment decision of where to go is derived from a continuous, mathematical formulation.

We found that, in general, the reactive rules were easier to design. Most rule-based strategies we designed seemed to work well in some cases, but invariably failed miserably under other conditions. In contrast, the reactive rules tend to react more reliably across a wide range of conditions. Furthermore, the use of numeric equations lends itself to running Monte Carlo simulations, and will eventually simplify the use evolutionary computing. In this section we describe in detail the reactive rule that seemed most robust and generally successful.

In our strategy, the evader employs simple reactive behavior with a small degree of planning. Its reactivity stems from the fact that it only possesses local information about its environment, a constraint that is meant to reflect the real-world constraint on robots. Specifically, the evader only knows about pursuers and navigation points that it can see, and has a very limited short-term memory about those it has seen previously. The evader's goal is to constantly seek out escape routes, and to flee from any pursuers it detects. Like pursuers, it navigates by means of navigation points, whose connections contain implicit structural information about the environment, and only reevaluates its target direction when it approaches an intersection (a navigation point) or is being chased by a pursuer.

The evader possesses a simple representation of the world. At each moment it can (but does not necessarily) choose from two or more *directions*, in which direction is loosely defined as a heading corresponding to a unique path or hallway to explore. When the evader is near a navigation point, the directions correspond to the neighbors of that point in the navigation graph. When the evader is on a link between points, there are always two directions, pointing towards the respective endpoints. For instance, the screenshot in Figure 1 shows the evader on a link between navigation points 21 and 22.

The evader decides where to go by calculating a *attractiveness* for each direction and following the one with the greatest value. The attractiveness of a direction is expressed as the sum of the attractiveness of every *branch* in that direction, where a branch is a visible navigation point containing at least one neighbor that is not visible. Branches are considered desirable because they offer possible escape routes. The attractiveness of a branch is based on several multiplicative factors: an *opportunity factor*, a *distance factor*, a *pursuer factor*, and an *angle factor*.

The first of these, the opportunity factor, is proportional to the number of links connecting that navigation point to unseen neighbors, discounting those with a pursuer visibly on them. The idea is that the more points a branch leads to, the more opportunities there are to escape. The next three factors are negative, in the sense that they discount the opportunity offered by a branch due to its inherent danger. As such, they reduce the attractiveness of a branch.

To explain the distance factor, several quantities must be explained: d denotes the Euclidean distance to the branch, t denotes the time necessary to reach the branch, and s denotes speed. The subscript e indicates the given variable pertains to the evader and p to the nearest pursuer to the branch. The distance factor is then given by the expression $a / (d_e + a)$, in which a is a semi-saturation constant, representing the distance greater than which a branch becomes "too far". This function begins at 1 when $d_e = 0$ and decays to 0 geometrically, reaching .5 when $d_e = a$, and is particularly sensitive to differences in distance less than a. The distance factor ensures that nearby branches are preferred to distant ones.

Figure 1: Screenshot of the simulator, showing three pursuers (blue) and one evader (red) navigating a bounded urban environment.

As with the distance factor, the pursuer factor is also a function of distance, but in this case it is relative to the nearest pursuer's distance. First, we define the quantity $Dd = [d_p - t_b s_p]^+$, which estimates how close the pursuer will be to the branch point by the time the evader reaches it. Negative values imply the pursuer will reach there first, and are mapped to 0. The factor itself is then given by $Dd / (b + Dd)$, which is similar to the previous factor except that instead of beginning at 1 and decaying to 0, it begins at 0 and approaches 1. This time the function reaches .5 at b, a semi-saturation constant representing at what length a lead over the pursuer becomes "safe".

The final factor is the angle factor, which was introduced specifically to address the following loop-hole in the pursuer factor: If the evader is in the same hallway as a pursuer and is considering a branch far enough behind the pursuer, it will ignore the fact that even though it would theoretically reach the branch first, it would have to go *through* the pursuer in order to do it. Therefore, we define q to be the (smaller) angle between the vectors pointing from the branch to the pursuer and the branch to the evader and let the angle factor equal $q / (g + q)$, where g is a where angles become "safe".

As stated before, the four attractiveness factors are multiplicative, so if any of them become very small the entire attractiveness does as well. In a sense, each is a link in a chain: A branch with a pursuer near it becomes invalidated despite its proximity and number of escape routes, while a distant branch might be invalidated despite appearing safe and offering a similar number of routes. In addition to the attractiveness factors, however, some simple special-case reasoning is

used to evaluate branches. For instance, if the pursuer factor of the first branch in a direction is 0, indicating it is too dangerous, the rest of the branches are assigned a value of 0, because that initial branch is a necessary waypoint to reach them. Also, the fact that the evader does not re-evaluate its goal while on an edge between navigation points unless being actively pursued, implicitly biases it against the direction it came from; without this "inertia", it would typically hover between points at the location where their distance and opportunity factors balance out.

Simulation results

We have run hundreds of simulations under various conditions to arrive at the particular evader strategy that we described. In this section we show some quantitative results obtained with this strategy, and we compare it to the performance of an evader that moves along a random succession of points.

For all simulation results below, we used the simulated environment shown in Figure 1. At each condition we ran the simulation ten times with ten different random seeds, and calculated the average across these ten runs. Our goal was to show how the performance can be quantified, and how certain key parameters in the simulation affect the performance.

As a measure of performance, we chose the amount of time that it takes for the pursuers to catch the evader (lower times being better for the pursuers). We then varied three different aspects of the simulation: first, we tested the performance as a function of the number of pursuers (3, 5, 7 or 10); second, we varied the speed of the evader (1.1, 1.5 or 1.9 times the speed of the pursuers); finally, we tested the reactive strategy and the random strategy. Hence the results we show below are based on a matrix of 24 different conditions (all possible combinations of the above), each run at least 10 times.

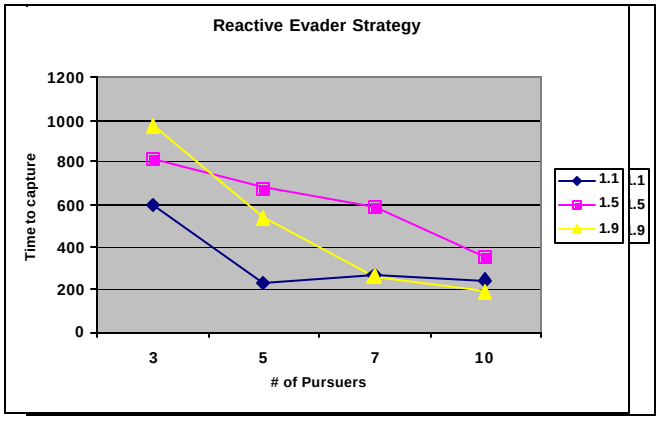

Figure 2: Performance of the random (left) and reactive (right) strategies as a function of the number of pursuers, and the relative speed of the evader. Performance is expressed as the time it takes to capture the evader.

Several points are evident from these simulations. First, and not surprisingly, in both cases (random or reactive strategies) the performance improves with the size of the blue team. What is

more interesting is that the improvement is not linear. This suggests that it should be possible to identify optimal strategies for allocation of UGVs based on mission parameters.

Second, and also fairly unsurprising, is the observation that in general the slower evaders are easier to catch. A less intuitive result is that with larger teams, the speed of the evader becomes less relevant. This, however, is due in large part to the fact that the environment is bounded: once the density of the pursuers becomes high, the evader simply has very few places to run away.

What seems impressive is that even a fairly simple-minded reactive evader strategy makes the pursuit task much more complicated. To emphasize the difference in time-to-capture between random and reactive, we reproduce in Figure 3 the two curves for when the evader has a speed of 1.5 times the speed of the pursuer.

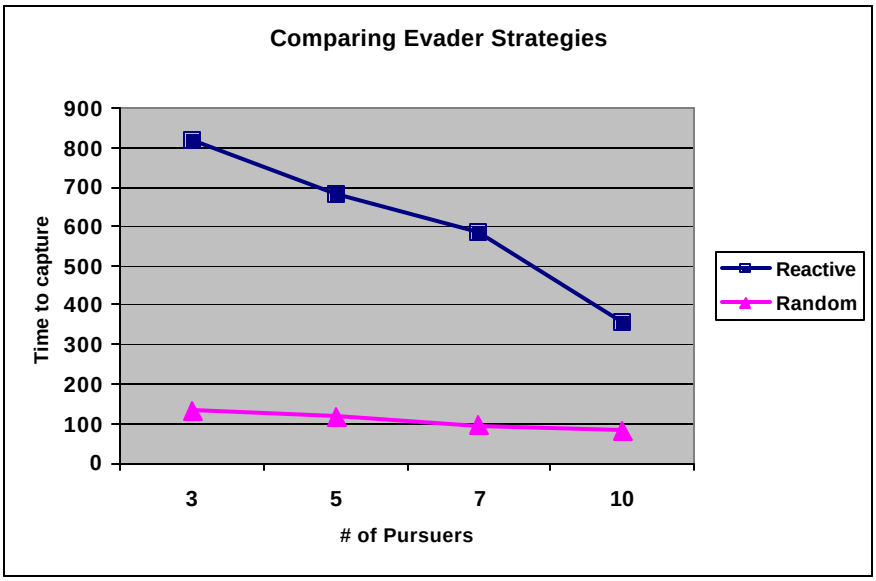

Figure 3: Comparison of the random and reactive strategies under identical conditions.

It is clear from this figure that the evader is much harder to catch: with the smallest blue team (3 pursuers), the evader is nearly eight times as hard to catch. Notice also that, because of the near lack of coordination between pursuers, the improvement resulting from larger teams is roughly linear: for instance, the 5-pursuer team takes almost exactly twice as long to capture the evader as a 10-pursuer team. We expect that a more powerful pursuer strategy would show a more dramatic improvement as the team size increases.

Future work

At this point we have completed the ground work for the project. The next step will be to use the software in conjunction with evolutionary algorithms to design more intelligent pursuer strategies. Several comments are in order.

First, particular attention must be paid to the nature of the rules and the representation used by the UGVs. For instance, the evader reactive rule could easily be improved by evolving some of the numerical factors that control the “attractiveness” equation. However, the current representation does not account for things such as memory of past locations. In the case of

pursuers, basic strategies will need to account for things such as a prediction of where the evader is heading, even when it is temporarily out of sight.

Second, it would be useful to derive a metric that describes the complexity of the environment. If we changed the environment dramatically, even if it was roughly the same size, the results might be quite different. An algorithm that automatically generates environments of a given size and complexity would be very useful.

Third, this problem lends itself very well to the application of *co-evolution*. Co-evolution refers to the process of allowing more than one type of "agent" to evolve. In this case we could co-evolve evader and pursuer strategies, with the aim of devising more sophisticated strategies than if we evolved each independently of the other. In addition, it should be possible to let the environment itself evolve: with a simple extension to our tool, we could devise an automatic environment generator that creates "legal" environments (that is, environments with no dead ends or unreachable places) of arbitrary complexity. The complexity of the environment could then evolve simultaneously with, for example, the pursuer strategy. This should result in pursuer strategies that can work robustly under a variety of conditions.

Fourth, this problem could also become a testbed for *interactive evolution*. This refers to a class of evolutionary methods in which humans are involved in the evolutionary process. In our case, we would like to transform the simulator into a Java applet, so that it could be run from any browser. Furthermore, we would allow users to "play" either as the pursuers or as the evader. In either case, their goal would be to beat the game: for instance, as pursuers they may have to catch the evader within a certain amount of time, and conversely as evaders. The data collected from the on-line playing would then be used off-line to evaluate the fitness of different computer strategies. This is a powerful way to generate large amounts of data in a realistic setting. A member of Icosystem's staff (Dr. Pablo Funes) has prior experience with this type of on-line evolution: he created an applet in which players tried to beat the computer at a simple version of the old video game *Tron*. His results were quite good, and we expect that the approach would be quite successful for our evader-pursuer simulation.

Finally, our ultimate goal would be to transfer the results obtained in simulations to a team of real robots. This would require some additional detail in the sensors and dynamics of individual agents, though we expect that the level at which we are modeling is sufficiently high that most results should be directly applicable.